

|  |
| --- |
|  |
| Mobile Authentication Corporation |
| OTP Services API, Version 1.6 |

|  |
| --- |
|  |

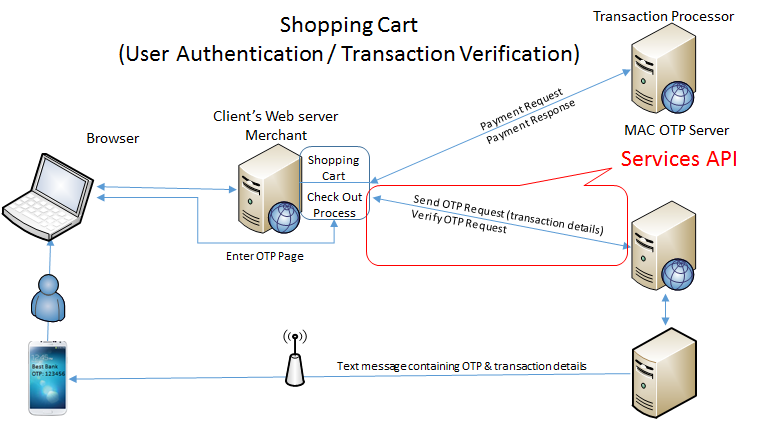
# Introduction

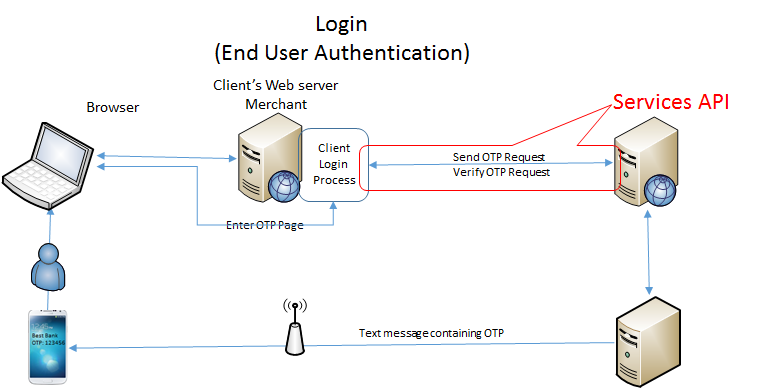
This document defines the API for the Web Services for the MAC OTP System (“System”). The System is comprised of several Web Services (“Services”) that provide the functionality needed to send and verify One-Time Passwords as text messages using the SMS network. The API also supports sending regular text messages (containing no OTP). As a configuration option, the System can send messages via the voice network. The details of the MAC OTP System Registration Service APIs are not included in this document.

Overview

Contained in this document are brief descriptions of the HTTP/HTTPS methods for sending requests to the Services, the responses, the message formats, and the client requirements. The document also contains JavaScript/JQuery examples of the functions which create and issue the calls, as well as, process the responses.

## System diagrams





Client Requirements

The MAC system requires every client to be registered with the system. When registered, the client will be issued a ***Client ID***. Every request issued to one of the Services must contain a valid Client Id. Optionally, the system supports the concept of a **Group.**  A group is a collection of one or more clients.

**Note**: Examples in this document do not have Group Ids.

## End Users

In the context of the MAC OTP System and this document “End Users” are the customers of the Clients. The end users are the people that receive the OTP messages on their SMS enabled devices. The System expects end user to be controlled by the client or be registered with the System. In the “Client Controlled” API calls it is the responsibly of the client to insure that the end user’s phone number and email address are valid. The System only checks the client id and the format of the phone number and email address. In the “Registered” API calls the end user must be registered before the OTP requests are processed by the System. The phone number and the email address are maintain in the System database. The Service API requires that the caller hash, using MD5 hash algorithm the unique user identification and user last name in order to create a unique ”UserId”. The System checks for a valid UserId before processing an OTP request.

## OTP Services

There are two Services that support the One-Time Password and send Message functionality, the RequestOTP Service and the VerifyOTP Service. Each has their own URL. The base URL (where the Services are running) combined with the Service URL make up the HTTP address.

RequestOTP: Otp/RequestOtp.asmx/WsRequestOtp

VerifyOTP: Otp/ValidateOtp.asmx/WsValidateOtp

## Service API Requests

Services support the HTTP/HTTPS Post method where the data contains the request details. Depending on the implementation, the data is either converted to hexadecimal strings or encrypted (recommended when using HTTP) before sending as data to the request, this avoids special character conflicts.

## Request Format Details

The parameters for a request are assembled in an ASCII string as a key value pairs with each key/value separated by the pipe character “|” and the keys are separated from the values by the colon character “:”. Some values, such as the Transaction details may contain special character that would cause problems in the request process. These values must be converted to a hexadecimal string before it is added to the request parameters. (See the coding examples section for details).

**Note**: The following example is for a “Client Managed End User” request. The client is responsible for supplying the end user’s phone number and email address. MAC’s OTP system does not maintain or verify the end user information. For details of a “Registered End User” request refer to the coding examples.

**Note**: This request has the Ad Pass Opt-out option set to do not send Ad. See the Ad Pass section for more details.

* Example of JavaScript function call

function RequestOtpClientManagedEndUser(

pClientId, // Client Id (required)

pEndUserPhoneNumber, // End user’s phone number (required, format is validated)

pEndUserEmail, // End user’s email address(required, format is validated)

pEndUserIp, // End user’s machine’s IP address (optional)

pTransactionType, // OTP Message type (optional, default is 0)

pTransactionDetails, // Transaction Details (optional, included in OTP message)

pAdPass, // Ad Pass Option (optional), see the Ad Pass section for details

pCallbackFunction)

* Example before hexadecimal encoding:

**Note**: Key value pairs with keys in red and values in blue (key value separator is in black).

Request:SendOtp|CID:5351674c74846919ec735074|PhoneNumber:4802684076|EmailAddress:tdavis@mobileauthcorp.com|EndUserIpAddress:192.168.168.1|TrxType:2|TrxDetails:4861742031372e39397c4a61636b657420243135302e39387c546f74616c20243136382e3937|APOpt:AdDisable|API:JS

Where:

1. The request (required): Request:SendOtp

**Note**: In this example send OTP to a Client managed end user.

1. Client Id (required): <CID:5351674c74846919ec735074>
2. End user’s mobile phone number (required): PhoneNumber:4802684076
3. End user’s email address (required): EmailAddress:tdavis@mobileauthcorp.com
4. End user’s machine IP address (optional): EndUserIpAddress:192.168.168.1
5. Transaction type (optional default is 0 ‘OTP’): TrxType:2
6. Transaction details (optional, default is no details in OTP message): TrxDetails:4861742031372e39397c4a61636b657420243135302e39387c546f74616c20243136382e3937

**Note 1**: Transaction details (the value) is hexadecimal encoding to avoid issues with special character

**Note 2**: see transaction details encoding for formatting details.

1. Ad Pass Option (Optional) if included in request data the client/user can choose to Opt-out (don’t send and Ad on the request).
   1. Set to "AdDisable" system will not send an Ad for this request regardless of client’s configuration.
   2. Set to "ADEnable" or not specified Ad will be sent according to the Client’s configuration.
   3. Who is making the request (optional, used for resolving errors): API:JS

* Example after hexadecimal encoding (complete data packet):

Data=

* Break down of components:

Http post header: Data=

Hexadecimal encoded indicator: 99

Length of client id: 24

Client id (as issued by MAC): 5351674C74846919EC735074

Request data (Hexadecimal encoded): 

### Transaction Type (TrxType) Encoding

The transaction type parameter is used by the Send OTP function to select the text message formatting template.

The templates are setup in when the client is registered with the system.

Transaction types are:

0: (TrxType:0) is for *notification* message “no OTP will be generated or included in message”,

1: (TrxType:1) is for *authentication* nominally used in the login process “no transaction details”,

2: (TrxType:2) is for *transaction verification* normally includes transaction details that get passes as a hexadecimal encoded string. The message assembly function decodes and formats based on the template.

### Transaction Details Format and Encoding

The transaction details could contain new lines and characters that can’t be sent as ASCII characters.

1. The new lines in the transaction details and in the send message text must be replaced by the pipe character “|”. The massage assembly logic replaces the pipe character with the appropriate new line sequence for the message delivery channel.
2. The transaction details and the send message body are hexadecimal encoded.

* Example before encoding:

Hat $17.99|Jacket $150.98|Shirt $33.98|Total $202.95

**Note:** The example will be displayed in the OTP message as 4 lines.

## Response Formats Details

All the OTP Services responses are formatted in XML. Responses contain a Reply node if successful and an Error node if unsuccessful.

Note: See Ad Pass section for details about how to process the Ad node in the response.

* Example of response to successful request:

<?xml version="1.0" encoding="utf-8"?>

<macResponse totalProcessTime="14ms">

<calledMethod>WsRequestOtp()</calledMethod>

<Reply>RequestId:53d1dc5174846958ac625de6|DeliveryMethod:Sms</Reply>

<RequestId>53d1dc5174846958ac625de6</RequestId>

<Add>

<Details>SendOtp:Kohl's</Details>

</macResponse>

**Note:** In the example above the response is for a “SendOTP” request. The RequestId is used as the correlation number and must be returned in the VerifyOTP request.

**Note:** The example response contains an Ad, see the Ad Pass section for more details on how to process the Ad data in the response.

* Example of response to unsuccessful request:

<?xml version="1.0" encoding="utf-8" ?>

<macResponse>

<Error>Invalid Request Id</Error>

</macResponse>

## Ad Pass

This section explains how the Ad Pass feature works, how to use the Ad Pass Option in the request and how to process that Ad Pass data in the response. The Ad returned in the response contains a Javascript code snippet designed to be inserted in the “Enter OTP” page. Preferably in a predefined <div> on the Enter OTP page.

### Overview of how Ad Pass Works

The Ad Pass feature processes 2 Ads for a SendOTP request, 1 Ad gets included in the OTP Text Message and other Ad get returned in the response. Both Ads are preconfigured in the Ad Server by the Client via the Ad Campaign process.

Note: The Ad Campaign process will be covered in a separate document.

When the Client calls the MAC OTP system to send an OTP to an end user and all the configuration and Opt-out options for send Ads to the end user are met, the OTP Server send a request to the Ad Server requesting an Ad for the client/user. The response from the Ad Server contains both Ads and the OTP server includes the “Message” Ad in the OTP Text message that gets sent to the end user mobile phone. The Otp server encodes the other Ad and includes the value in the RequestOTP response. The Client decodes the Ad an inserts the Ad into the Enter OTP page.

### Processing (Request)

The Ad Pass feature has several options that control the processing,

1. The in the Client’s configuration the Ad Pass option must be set. The is done via the Administrative interface.
2. If using Registered user OTP request the end user’s configuration must have the Ad Opt-out flag set to false. This is done when the user is first registered or through the End User Management service, not covered in this document.
3. The Ad Pass Option ”APOpt” is included in the request,
   1. Set to "AdDisable" system will not send an Ad for this request regardless of client’s configuration.
   2. Set to "ADEnable" or not specified Ad will be sent according to the Client’s configuration.
4. The Client has registered with an Ad portfolio.

### Processing (Response)

The Ad in the response is a string that is hexadecimal encoded and must be decoded to a string before it is inserted into the Enter OTP Page.

Example of decoded Ad from the Ad demo server site:

<div data-ad-id="bb9e7016-635a-40ac-83ef-448293b7cf7d"><a target="\_blank" href="http://crm.managedlogix.com" class="ads-link"><img src="http://api.ads.com\Demo/Data/Images\C370C5BA-E1C9-4491-BDD0-2F1848473062.png" width="392" height="72" border="0"></a><script type="text/javascript" src="http://api.ads.com/tracking.js"></script></div>

# Detail Coding Examples

The following examples are provided as JavaScript/JQuery source and intended to give the integrator the details need to implement the various OTP Service calls.

//-------------- One Time Password service calls -----------------------------------

//-------- Send OTP Message to a client managed end user ---------------------------

function RequestOtpClientManagedEndUser(

pClientId, // Client Id (required)

pGroupId, // Group Id (optional)

pEndUserPhoneNumber, // End user’s phone number (required, format is validated)

pEndUserEmail, // End user’s email address(required, format is validated)

pEndUserIp, // End user’s machine’s IP address (optional)

pTransactionType, // OTP Message type (optional, default is 0)

pTransactionDetails, // Transaction Details (optional, included in OTP message)

pAdPass, // Ad Pass Option, see the Ad Pass section for details

pCallbackFunction)

{

var requestData = "Request:SendOtp"; //Command to service

if (pClientId.Length == 0) //Client Id as issued by MAC

return ("Client ID required!");

requestData += "|CID:" + pClientId;

if (pGroupId != 0) // Optional if client request is restricted to a group

requestData += "|GroupId:" + pGroupId;

if (pEndUserPhoneNumber.Length == 0)

return ("End User's Phone Number required!");

requestData += "|PhoneNumber:" + pEndUserPhoneNumber;

if (pEndUserEmail.Length == 0)

return ("End User email address required!");

requestData += "|EmailAddress:" + pEndUserEmail;

if (pEndUserIp.Length != 0)

requestData += "|EndUserIpAddress:" + pEndUserIp;

if (pLabelIndex.Length != 0) {

requestData += "|TrxType:" + pLabelIndex;

}

if (pTransactionDetails.Length != 0)

requestData += "|TrxDetails:" + StringToHex(pTransactionDetails);

if (pAdPass != 0) // Optional Client is setting Ad Pass option

requestData += "|APOpt:" + pAdPass; // ‘APEnable’ send Ad, ‘APDisable’ do not send Ad.

requestData += "|API:JS"; // who is calling service

// 99 indicates the data is converted to a hex string (not encrypted)

var data = "Data=99" + pClientId.length.toString() + pClientId.toUpperCase() +

StringToHex(requestData);

$.post(GetMACServicesBaseURL() + RequestOtpWebService, data, pCallbackFunction);

return "";

}

//-------- Send OTP Message to a Registered end user ---------------------------

function RequestOtpRegisteredEndUser(

pClientId, // Client Id (required)

pGroupId, // Group Id (optional)

pEndUserUniqueIdentifier, // End user’s unique identifier, as registered (required)

pEndUserLastName, // End user’s last name, as registered (required)

pEndUserIp, // End user’s machine’s IP address (optional)

pTransactionType, // OTP Message type (optional, default is 0)

pTransactionDetails, // Transaction Details (optional, included in OTP message)

pAdPass, // Ad Pass Option see Ad Pass section later in this document

pCallbackFunction)

{

var requestData = "Request:SendOtp"; //Command to service

if (pClientId.Length == 0) //Client Id as issued by MAC

return ("Client Id required!");

requestData += "|CID:" + pClientId;

if (pGroupId != 0) // Optional if client request is restricted to a group

requestData += "|GroupId:" + pGroupId;

if (pEndUserUniqueIdentifier.length == 0)

return ("End User UID required!");

if (pEndUserLastName.length == 0)

return ("End User Last Name required!");

// Unique UserId: Must match the UserId used in the registration process

requestData += "|UserId:" + HashUserId(pEndUserLastName, pEndUserUniqueIdentifier)

if (pEndUserIp.Length != 0)

requestData += "|EndUserIpAddress:" + pEndUserIp;

if (pLabelIndex.Length != 0)

requestData += "|TrxType:" + pLabelIndex;

if (pTransactionDetails.Length != 0)

requestData += "|TrxDetails:" + StringToHex(pTransactionDetails);

requestData += "|API:JS"; // who is calling service

// 99 indicates the data is converted to a hex string (not encrypted)

var data = "Data=99" + pClientId.length.toString() + pClientId.toUpperCase() +

StringToHex(requestData);

$.post(GetMACServicesBaseURL() + RequestOtpWebService, data, pCallbackFunction);

return "";

}

//------------------ Verify an OTP ---------------------------

function VerifyOtp(

pClientId, // Client Id (required, must be the same client Id that made the send OTP request)

pRequestId, // OTP Id (required, returned by the Send OTP request)

pOtp, // OTP (required, entered by the end user)

pCallbackFunction)

{

var requestData = "Request:VerifyOtp"; //Command to service

if (pClientId.Length == 0) //Client Id as issued by MAC

return ("Client ID required!");

requestData += "|CID:" + pClientId;

if (pRequestId.Length == 0)

return ("Request ID required!");

requestData += "|RequestId:" + pRequestId;

if (pOtp.Length == 0)

return ("Otp required!");

requestData += "|OTP:" + pOtp;

requestData += "|API:JS"; // who is calling service

// 99 indicates the data is converted to a hex string (not encrypted)

var data = "Data=99" + pClientId.length.toString() + pClientId.toUpperCase() +

StringToHex(requestData);

$.post(GetMACServicesBaseURL() + VerifyOtpWebService, data, pCallbackFunction);

return "";

}

//------------------ Cancel an OTP ---------------------------

function CancelOtp(

pClientId, // Client Id (required, must be the same client Id that made the send OTP request)

pRequestId, // OTP Id (required, returned by the Send OTP request)

pCallbackFunction)

{

var requestData = "Request:CancelOtp"; //Command to service

if (pClientId.Length == 0) //Client Id as issued by MAC

return ("Client ID required!");

requestData += "|CID:" + pClientId;

if (pRequestId.Length == 0)

return ("Request ID required!");

requestData += "|RequestId:" + pRequestId;

requestData += "|API:JS"; // who is calling service

// 99 indicates the data is converted to a hex string (not encrypted)

var data = "Data=99" + pClientId.length.toString() + pClientId.toUpperCase() +

StringToHex(requestData);

$.post(GetMACServicesBaseURL() + RequestOtpWebService, data, pCallbackFunction);

return "";

}

//-------- Resend OTP Message to an end user ---------------------------

function ResendOtp(

pClientId, // Client Id (required, must be the same client Id that made the send OTP request)

pRequestId, // OTP Id (required, returned by the Send OTP request)

pCallbackFunction)

{

var requestData = "Request:ResendOtp"; //Command to service

if (pClientId.Length == 0) //Client Id as issued by MAC

return ("Client ID required!");

requestData += "|CID:" + pClientId;

if (pRequestId.Length == 0)

return ("Request Id required!");

requestData += "|RequestId:" + pRequestId;

requestData += "|API:JS"; // who is calling service

// 99 indicates the data is converted to a hex string (not encrypted)

var data = "Data=99" + pClientId.length.toString() + pClientId.toUpperCase() +

StringToHex(requestData);

$.post(GetMACServicesBaseURL() + RequestOtpWebService, data, pCallbackFunction);

return "";

}

//------------- Send Text Message service calls -------------------------------------

//-------- Send Text Message to a client managed end user ---------------------------

function SendMessageToClientManagedEndUser(

pClientId, // Client Id (required)

pGroupId, // Group Id (optional)

pEndUserPhoneNumber, // End user’s phone number (required, format is validated)

pEndUserEmail, // End user’s email address(required, format is validated)

pEndUserIp, // End user’s machine’s IP address (optional)

pMessage, // Body of text message (pipe characters for new lines)

pCallbackFunction)

{

var requestData = "Request:SendMessage"; //Command to service

if (pClientId.Length == 0) //Client Id as issued by MAC

return ("Client ID required!");

requestData += "|CID:" + pClientId;

if (pGroupId != 0) // Optional if client request is restricted to a group

requestData += "|GroupId:" + pGroupId;

if (pEndUserPhoneNumber.Length == 0)

return ("End User's Phone Number required!");

requestData += "|PhoneNumber:" + pEndUserPhoneNumber;

if (pEndUserEmail.Length == 0)

return ("End User email address required!");

requestData += "|EmailAddress:" + pEndUserEmail;

if (pEndUserIp.Length != 0)

requestData += "|EndUserIpAddress:" + pEndUserIp;

if (pMessage.Length == 0)

return ("Message required!");

requestData += "|Message:" + StringToHex(pMessage.replace(/\n/g, "|"));

requestData += "|API:JS"; // who is calling service

// 99 indicates the data is converted to a hex string (not encrypted)

var data = "Data=99" + pClientId.length.toString() + pClientId.toUpperCase() +

StringToHex(requestData);

$.post(GetMACServicesBaseURL() + RequestOtpWebService, data, pCallbackFunction);

return "";

}

//-------- Send Text Message to a registered end user ---------------------------

function SendMessageRegisteredEndUser(

pClientId, // Client Id (required)

pGroupId, // Group Id (optional)

pEndUserUniqueIdentifier, // End user’s unique identifier, as registered (required)

pEndUserLastName, // End user’s last name, as registered (required)

pEndUserIp, // End user’s machine’s IP address (optional)

pMessage, // Body of text message (pipe characters for new lines)

pCallbackFunction)

{

var requestData = "Request:SendMessage"; //Command to service

if (pClientId.Length == 0) //Client Id as issued by MAC

return ("Client ID required!");

requestData += "|CID:" + pClientId;

if (pGroupId != 0) // Optional if client request is restricted to a group

requestData += "|GroupId:" + pGroupId;

if (pEndUserUniqueIdentifier.length == 0)

return ("End User UID required!");

if (pEndUserLastName.length == 0)

return ("End User Last Name required!");

// create unique UserId using md5 hash

requestData += "|UserId:" + HashUserId(pEndUserLastName, pEndUserUniqueIdentifier)

if (pEndUserIp.Length != 0)

requestData += "|EndUserIpAddress:" + pEndUserIp;

if (pMessage.Length == 0)

return ("Message required!");

requestData += "|Message:" + StringToHex(pMessage.replace(/\n/g, "|"));

requestData += "|API:JS";// who is calling service

// 99 indicates the data is converted to a hex string (not encrypted)

var data = "Data=99" + pClientId.length.toString() + pClientId.toUpperCase() +

StringToHex(requestData);

$.post(GetMACServicesBaseURL() + RequestOtpWebService, data, pCallbackFunction);

return "";

}

//-------- Create UserId using user’s last name and Unique Identifier -------------------

function HashUserId(pLastName, pUniqueIdentifier) {

return hex\_md5(pLastName.toLowerCase() + pUniqueIdentifier.toLowerCase()).toUpperCase();

}

# Change History

V1.1 - Original document

V1.2 – Update request parameter names “ToPhone” to “PhoneNumber” and “ToEmail” to “EmailAddress”, add some comments.

V1.3 – update RequestOtpRegisteredEndUser SendMessageRegisteredEndUser examplet where the UserId is created by calling a MD5 hash function, included the HashUserId function to show the end user and the Unique Identifier “UID” are set to lower case before doing the MD5 hash.

V1.4 – Add Ad Pass Opt-out option to RequestOtpClientManagedEndUser example.

V1.5 – Add section on TrxType,

in VerifyOtp request the OTP key is upper case,

Update Response for SendOtp Request.

V1.6 – Ad Pass details added to RequestOTP response